**Earth Observing System**
**Geoscience Laser Altimeter System**

# GLAS I-SIPS Software Delivery Package

**Version 0**

**July, 1999**


**Contact:**

| | |
|---|---|
| **Anita Brenner** | **anita@icesat2.gsfc.nasa.gov** |
| **David Hancock** | **hancock@osb1.wff.nasa.gov** |
| **Gladstone Marcus** | **marcus@icesat2.gsfc.nasa.gov** |
| **Jeff Lee** | **jlee@osb3.wff.nasa.gov** |

# Version 0 Delivery Package for GLAS I-SIPS Software



## Introduction

The GLAS I-SIPS software consists of a Science processing system that creates multiple data products, A Data Preparation and handling subsystem that supports the Science Processing system by acquiring raw, input data and user information to perform scheduling , staging, archiving and distribution of the data products. Finally, a set of utilities provide the capability for formatting the data products, creating intermediate ancillary products and, generally, supporting quality control.
For Version 0 provides a minimum set of functionalities as specified in the Version 0 requirements and is intended to be the basis for development and delivery of the full set of functionalities, to be provided as Version 1, that will satisfy all the requirements set out in the Architecture document.

The Version 0 science processing system accomplishes the execution of 4 scenarios, listed below as “Specific Requirements”, that are required for the software, and depends on user provided control files to specify the execution parameters.

The data preparation and handling capability that is delivered for Version 0 consists of ported software that performs archiving and retrieval of data using the DLT tape media. The ported software consists of legacy code from the Version 0 DAAC known as Archer. The software and its location are described in the document:
“Data Preparation and Handling: Ported GSFC DAAC V0 Software for Version 0 Delivery.”

## Requirements for v0 Delivery

The basic requirement for version 0 is to have a framework in place that addresses all external interfaces. We will be somewhat limited on this because of the late delivery of the telemetry descriptions, the ATBDs, and the understanding of the "DAAC version 0" software.

*General requirements:*

- All standard data products in an SCF format are produced.
- Several normal processing strings execute without errors.
- All sub ATDB version 0 interfaces are exercised.
- Processing history will be available.
- Formalization for standardized messaging and error-handling in Local ancillary files will be incorporated
- A working version of I-SIPS archer.
- Draft documentation on using the s/w delivered.
- Software will read those ANC files which have been defined.

*Specific requirements:*

- Control file processing.
- Initialization file processing.
- One processing string to create GLA01 to GLA15.

- One processing string that starts with a GLA05 input to produce GLA06, 12,13,14,15.
- One processing string that starts with GLA02 input and produces GLA07,08,09,10,11.
- One processing string that starts with a GLA05 input to produce GLA06.
- Each Sub ATDB receives known data and returns known data onto associated output product(s).
- Defined set of version 0 local ANC files.

*Acceptable Requirements not fully delivered*

- Handling of missing data
- True time tag conversion and alignment
- Ancillary data file inputs and outputs
- QA data and processing.
- Anc files from external sources
- Archive information

## Assumptions for v0 Delivery

- There will be a 1-1 temportal correspondence between L1A and L2 GLA_SCF files. This means that the v0 software will not use multiple files of the same File Type for processing or reprocessing. This capability, however, will be built into the control structures for future use and some of this functionality is already present in the V0 release.
- All input data will be time aligned with no missing data or error data.
- Imported ANC files will not be available.
- The GLA_ANC_06 file will not be created using the Toolkit.
- GLA_SCF files will be raw data only. No header or metadata is required.
- GLA00 data will be time-aligned and all APIDs merged into a single binary file.

## System Design Overview

The I-SIPS data processing system is designed to be both efficient and flexible. The system was designed for operational flexibility considering data availability constraints and reprocessing requirements. In order to meet these requirements, the design of the I-SIPS data processing system is based on three software layers of increasing intelligence. The bottom layer is composed of the implementations of scientific algorithms, the middle layer is composed of multiple managers responsible for the execution of the scientific algorithms and data output, and the top layer is a main program which is responsible for data input and the execution of each manager. The core of the I-SIPS design is a consolidation of necessary scientific algorithms. These algorithms are published in Algorithm Theroretical Basis Documents (ATBD) provided by the GLAS Science Team. The provided algorithms are grouped into ATBD subsystems separated by scientific discipline. The four subsystems are: L1A, Waveforms, Atmosphere, and Elevation. Additionally, the subsystems are designed such that data required by each subsystem is available from a product (data file) written by a preceding subsystem. The result of is no (or very little) data dependence between the subsystems.

Associated with the ATBD subsystem are corresponding Subsystem Managers. The Subsystem Managers control which sub-ATBD algorithms to execute and what data products get written. Very little intelligence is built into the Subsystem Managers except for the order in which sub-ATBD algorithms are called and what is required to write each data product.
Finally, surrounding the four managers is a main program that is basically a state machine. This program uses control input to determine what data to read, what processes to execute, and what data to write.
During the design phase, both single-program and multiple-program approaches were considered. Advantages of the single program approach include a potentially reduced I/O burden (dependent on the operational use of the software), less software maintainence, and ease of configuration management. The disadvantages of this approach are slightly more code complexity and a potential for increased memory usage due to non-active global memory being used. The advantages of multiple programs are simplicity and relatively reduced memory usage. The disadvantages are an increased software maintenance burden and increased configuration management difficulty.
By keeping the top-level code very generic and developing on the single-program approach, the Team retains maximum development flexibility. If there is a solid reason not to implement the I-SIPS data processing system as a single program, it will be very easy to subset GLAS_Exec and create four independent programs. This will allow the Team to take advantage of new information as development progresses into prototyping and load testing with a minimum of wasted effort.
The team recognizes that there will be other task–specific software which will interface with data created by the I-SIPS data processing system. In order to effect the reuse of this software, the GLAS Team will implement major components and subsystems as shared libraries. These libraries (especially the I/O library) will be as generic as possible so they may be used without modification by both internal and external programmers.

## Data Management Interface Requirements

The design of GLAS_Exec imposes the following requirements on the GLAS Data Management System (DMS):

- The DMS will retrieve and stage into a directory all input files necessary for the requested execution.
- The DMS scheduler will create GLAS_Exec's control file via a predefined recipe. This includes the systematic naming of output files.
- The DMS scheduler will invoke the GLAS_Exec execution instance with the control file name as an argument.
- The DMS scheduler will handle a return code from GLAS_Exec specifying either success or failure. Additionally, if a fatal error occurs during the GLAS_Exec initialization, text may be written to standard output.

## GLA00 Definition

The V0 version of GLAS_Exec is coded using a modified version of the GLA00 definition Version 2.0. The modifications are documented and were necessary due to misalignment of data structures within the ANC_SCI Apid.

V0 GLAS_Exec will read GLA00 data which are contained in a single file of variable length records. Data are time-aligned and delimited by WFF-designated Apids containing the record index for the delimited data.

## Inconsistency between GLA00 and GLA02

Since we did not update the other GLAxx products when we adopted the GLA00 V2.0 definitions, there is an inconsistency with the number of –1 to 10km samples. In various places there are either 132 or 133 samples. The correct is 132. Internally, GLA00 is now consistent. However, GLA02 has not been fixed since we froze v0 on the definitions previously provided.

The same problem exists for the number of samples in 40km to 20km data. GLA02 uses 267, GLA00 uses 268.

## Initial Error Handling

Since input file names are designated in the control file and GLAS_Error is initialized by ANC07, GLA06 and GLAS_Error will be unavailable to the initialization and control-file parsing portions of program execution. For this part of the run, Error messages will be printed to standard output and error severity determined internally. If desired, the scheduler can redirect GLAS_Error's stdout into a log file for later analysis. This method provides maximum flexibility for interaction between GLAS_Exec and the scheduler in early program execution.
To do this, the global ANC06 file structure will be initialized to unit 6 and named "stdout". When the Control file is parsed and the specified ANC06 file is opened, the global ANC06 will be updated accordingly.

Modules which use stdout for error handling include:
GetControl_mod
OpenFiles_mod
OpenFInFile_mod
OpenFOutFile_mod
OpenCInFile_mod
OpenCOutFile_mod

## Time/Record Synchronization

GLA00 data time/record synchronization between APIDs is currently TBD.
For GLA01 and higher-level products, a *RecIndex* field will be used for record synchronization. This field will be present in each GLA file record and will contain a number that will identify each 1/sec GLAS science measurement. The current definition is as follows:

YYDDDSSSSS
where
YY= number of years from 2000
DDD= day of year
SSSSS= unique second of day
In the case of missing data, the number will be incremented to account for the number of records expected.

## File Naming Conventions

GLAS GLA files will be named as follows:
**GLA*xx_ccc_tttt_s.dat*** *(for SCF-type file)*
**GLA*xx_ccc_tttt_s.hdf*** *(for HDF-type file)*
where xx = Product ID
ccc = Cycle (000-999)
tttt = Track (0000-2600)
s = Segment, (0=none, 1-4 correspond to 50° lat/lon breaks)

GLAS ANC files will be named as follows:

**ANC*xx_TBD.dat***
where xx = Product ID
TBD = To Be Determined

These conventions fulfill the requirement that the type of product is carried in the filename. It also contains temporal and geographic information which allows for a gross-level data selection.

## Granule Size

Ignoring GLA00, GLA01 is the minimum sized granule; other granules are multiples of GLA01. Corresponding output files will be opened when a new GLA01 is read. Granules never break except on a GLA01 transition.

| Prod ID | Rec Size (bytes) | Mbytes / Day | Rate / sec | Num Recs | granules | | | Notes |
|---|---|---|---|---|---|---|---|---|
| | | | | | /rev | /day | size | |
| GLA01 - land | 4020 | 1129 | 1 / 1 | 7 | 4 | 56 | 30 | land 46.6%; |
| - ocean | 4020 | 552 | 1 / 1 | 3 | | | | ocean 53.4% |
| GLA02 | 28652 | 2475.5 | 1 / 1 | 1 | 0.5 | 7 | 354 | |
| GLA03 | 1000 | 86.4 | 1 / 1 | 1 | 0.07 | 1 | 86 | estimate |
| GLA04 | 28000 | 2419.2 | 1 / 1 | 1 | 0.14 | 2 | 1210 | estimate |
| GLA05 | 9636 | 832.6 | 1 / 1 | 1 | 4 | 56 | 15 | |
| GLA06 | 5236 | 452.4 | 1 / 1 | 1 | 0.07 | 1 | 452 | |
| GLA07 | 63836 | 5515.4 | 1 / 1 | 1 | 0.5 | 7 | 788 | |
| GLA08 | 780 | 3.4 | 1 / 20 | 1 | 0.07 | 1 | 3 | |
| GLA09 | 2948 | 63.7 | 1 / 4 | 1 | 0.07 | 1 | 64 | |
| GLA10 | 13812 | 298.3 | 1 / 4 | 1 | 0.07 | 1 | 298 | |
| GLA11 | 568 | 12.3 | 1 / 4 | 1 | 0.07 | 1 | 12 | |
| GLA12 | 3004 | 29.1 | 1 / 1 | 1 | 0.07 | 1 | 29 | Antartic suface only |
| GLA13 | 4076 | 17.6 | 1 / 1 | 1 | 0.07 | 1 | 18 | Guesstimate |
| GLA14 | 5416 | 165.7 | 1 / 1 | 1 | 0.07 | 1 | 166 | Includes all surface except ocean or antartic |
| GLA15 | 2888 | 133.2 | 1 / 1 | 1 | 0.07 | 1 | 133 | |

## Control File Format

The GLAS Control File will be dynamically generated by the Data Management scheduler and redirected into the GLAS_Exec process. The Data Management system will have a database of “recipes” which are used to create unique control files for pre-defined operational scenarios. The standard method of control will be the control file. It may be generated by the database or by hand. The control file is passed as a command line argument to GLAS_Exec. The secondary method of control occurs when GLAS_Exec detects that there is no control file argument passed on the command line. When no file is specified, GLAS_Exec runs through an interactive text-based interface which has the same options that could be specified by the control file. The dual-control method allows for a both tightly-controlled standard processing and easily customized special-case processing.

Like the GLA_ANC_06 and GLA_ANC_07 datafiles, the Control File is based on the ‘KEYWORD=VALUE’ construct. The construct consists of a line containing a keyword/value pair delimited by an equals sign (=). The ordering of the keywords is not relevant but should follow a convention for consistency. Multiple instances of certain keywords are allowed. Comments may be placed in the control file by prepending the comment with a # character. Lines should be limited to a maximum of 80 characters.

Required single-instance keywords include:

| | |
|---|---|
| TEMPLATE_NAME= | Name of the control file template. |
| EXEC_KEY= | Unique (per day) execution key |
| DATE_GENERATED= | Date the control file was generated. |

| | |
|---|---|
| OPERATOR= | Operator who generated the control file. |
| CYCLE= | Cycle of data |
| REV= | Revolution of data |

Required multiple-instance keywords include:

| | |
|---|---|
| INPUT_FILE= | Input file and version. |
| OUTPUT_FILE= | Output file and version |

Optional multiple-instance keywords include:

| | |
|---|---|
| SURFACE_TYPE= | Surface Type to Process (for all ATBDs) |
| L1A_PROCESS= | L1A Process to Execute |
| WAVEFORM_PROCESS= | Waveform Process to Execute (or scenario) |
| ATMOSPHERE_PROCESS= | Atmosphere Process to Execute (or scenario) |
| ELEVATION_PROCESS= | Elevation Process to Execute (or scenario) |

Filenames and versions included in the Control File are generated by the scheduler via a pre-defined recipe. File naming is currently TBD, but a requirement for naming is that output names may be uniquely generated from input names.
Additionally, pre-defined, subsystem-specific indentifiers may specify which processes are executed, rather than a verbose list of processes. The SURFACE_TYPE keyword specifies over what type of surface processing should occur. The default is all surfaces. Keywords and values are not case-specific (they will be converted to all lower case during parsing) but it is recommended that, for consistency, keywords be entered in upper case.

*Control File Template*

```
#
# These are comments which are skipped by the parser
#
CONTROL_FILE_NAME=      Name
EXEC_KEY=          Key
DATE_GENERATED=    Date
OPERATOR=          Name
CYCLE=             Cycle
REV=         Rev
INPUT_FILE=        Input_File  File_Version
OUTPUT_FILE=       Output_File File_Version
SURFACE_TYPE=      Surface Type
L1A_PROCESS=       L1A Process or Scenario
WAVEFORM_PROCESS=  WF Process or Scenario
ATMOSPHERE_PROCESS=     Atm Process or Scenario
ELEVATION_PROCESS=      Elev Process or Scenario
END-OF-CONTROL-FILE
```

Values for fixed-response keywords are listed below:

| Keyword | Values |
|---|---|
| SURFACE_TYPE | ALL (default) |
| | LAND |
| | OCEAN |
| | SEAICE |
| | ICESHEET |
| L1A_PROCESS | ALL |
| | NONE (default) |
| | L_Gen_ALT |
| | L_Gen_ATM |
| | L_Gen_ATT |
| | L_Gen_ENG |
| WAVEFORM_PROCESS | ALL |
| | NONE (default) |
| | W_Assess |
| | W_DetGeoSurTyp |
| | W_CalcOtherCh |
| ATMOSPHERE_PROCESS | ALL |
| | NONE (default) |
| | A_interp_pod |
| | A_interp_met |
| | A_mbscs |
| | A_cal_cofs |
| | A_ir_bscs |
| | A_g_bscs |
| | A_avg_bscs |
| | A_cld_lays |
| | A_pbl_aer_lays |
| | A_aer_lays |
| | A_cld_opt_prop |
| | A_aer_opt_prop |
| ELEVATION_PROCESS | ALL |
| | NONE (default) |
| | E_CalcLoadTD |
| | E_CalcOceanTD |
| | E_CalcEarthTD |
| | E_CalcPoleTD |
| | E_GetGeoid |
| | E_CalcTrop |
| | E_IntrpPOD |
| | E_CalcStdIR |
| | E_CalcLdIR |
| | E_CalcOcIR |
| | E_CalcSiIR |
| | E_CalcIsIR |
| | E_CalcStdSp |

Sample L1A-Only Control File

```
#
#Sample Control File which only runs L1A Processes
#
TEMPLATE_NAME=     L1A_AND_PARTIAL_WF_CONTROL_FILE
EXEC_KEY=          000012
DATE_GENERATED=    26-January-1999
OPERATOR=          jlee
CYCLE=             01
REV=         2000
SURFACE_TYPE=      ALL
INPUT_FILE=        anc04_01_2000_0.dat      2
INPUT_FILE=        anc05_01_2000_0.dat      1
INPUT_FILE=        gla00_01_2000_0.dat      1
OUTPUT_FILE=       anc06_01_2000_0.dat      2
OUTPUT_FILE=       gla01_01_2000_0.dat      2
OUTPUT_FILE=       gla02_01_2000_0.dat      2
OUTPUT_FILE=       gla03_01_2000_0.dat      2
OUTPUT_FILE=       gla04_01_2000_0.dat      2
L1A_PROCESS=       ALL
WAVEFORM_PROCESS=  W_ASSESS
WAVEFORM_PROCESS=  W_DETGEOSURTYP
END-OF-CONTROL-FILE
#
# End of Control File
#
```

## ANC06 Definition and Format

A GLA ANC 06 file will be generated for each execution of the GLAS_Exec. The file will contain processing information and status, error messages, QA data, and data required to generate the GLAS product metadata. The GLA ANC06 will be opened and initialized by GLAS_Exec. The GLA ANC 06 file is an ASCII file and the contents are in keyword = value format..

A full list of acceptable keywords will be documented at a later date. The following list is those keywords defined for the v0 delivery:

| | |
|---|---|
| VERSION= | Version of GLAS_Exec components |
| {All acceptable Control file keywords} | |
| ERROR= | Number, Type, Message |
| GLAS_EXEC_STATUS= | Type, Message |
| L1A_STATUS= | Type, Message |
| WF_STATUS= | Type, Message |
| ATM_STATUS= | Type, Message |
| ELEV_STATUS= | Type, Message |
| GLAS_EXEC_ SUMMARY = | Type, Message |
| L1A_SUMMARY= | Type, Message |
| WF_ SUMMARY = | Type, Message |
| ATM_ SUMMARY = | Type, Message |
| ELEV_ SUMMARY = | Type, Message |

Sample ANC06 File

```
VERSION              =           0 GLAS_EXEC V0.1
VERSION              =           0 PLATFORM_LIB V0
VERSION              =           0 CIO_LIB V0
VERSION              =           0 IO_LIB V0.2
VERSION              =           0 CNTL_LIB V0.6
VERSION              =           0 L1A_LIB V0.1
VERSION              =           0 WAVEFORM_LIB V0.1
VERSION              =           0 ATMOSPHERE_LIB V0.1
VERSION              =           0 ELEVATION_LIB V0.1
CONTROL              =           0 DATE_OF_RUN=29-March-1999 06:28:12
CONTROL              =           0
TEMPLATE_NAME=L1A_AND_PARTIAL_WF_CONTROL_FILE
CONTROL              =           0 EXEC_KEY=000012
CONTROL              =           0 DATE_GENERATED=26-January-1999
CONTROL              =           0 OPERATOR=jlee
CONTROL              =           0 CYCLE=01
CONTROL              =           0 REV=2000
CONTROL              =           0 SURFACE_TYPE=ALL
CONTROL              =           0 INPUT_FILE=gla00_01_2000_0.dat        1
CONTROL              =           0 INPUT_FILE=anc07_01_2000_0.dat        1
CONTROL              =           0 OUTPUT_FILE=gla01_01_2000_0.dat       2
CONTROL              =           0 OUTPUT_FILE=gla02_01_2000_0.dat       2
CONTROL              =           0 OUTPUT_FILE=gla03_01_2000_0.dat       2
CONTROL              =           0 OUTPUT_FILE=gla04_01_2000_0.dat       2
CONTROL              =           0 OUTPUT_FILE=anc06_01_2000_0.dat       2
CONTROL              =           0 L1A_PROCESS=ALL
EXEC_STATUS          =           0 GLAS_Exec Start of Processing
L1A_STATUS           =           0 L1A Started Processing
ERROR                =         123 2798 Frame checksum error
ERROR                =         124 1078 Previous GLA00 Record Missing
EXEC_STATUS          =        1500 End of Processing
```

GLAS_Exec will write the VERSION of software executed and CONTROL inputs to the GLA ANC 06 file. GLAS_Exec will write STATUS messages indicating start and end of processing and number of records processed.
Error messages generated during processing are written to the GLA ANC 06 file. The GLAS_Error subroutine will call the WriteANC06 subroutine. During wrap-up the total number of errors shall be written to the GLA ANC 06 file.
Since the GLA ANC 06 file is in ASCII it can be read with any text editor or word processor. Additionally users can use grep to search for keywords to generate reports of specific interest, i.e., ERROR reports. Standard scripts can be created that will generate subsets of the GLA ANC 06.
The ATBD Managers will write messages to GLA ANC 06 indicating start of processing.

## ANC07 Definition and Format

ANC07 is the constants file. It will be read to intialize those constants which may be changed without requiring a code change.
ANC07 will use the same keyword/value format as the Control file. A full list of acceptable keywords will be documented at a later date. A sample ANC07 file follows:

Sample ANC07 File

```
#
```

```
# GLAS v0 ANC07 Constants File
#
ANC07_VERSION = v0.1 02 May 1999
#
# Constants Mod Entries
#
BEG_OF_CONSTANTS_MOD = ----------------------------------------------------
--
#
# Set Number of Constants
#
NUM_CONSTANTS = 6
#
# GLA01 Constants
#
gi_GLA01_Main_ID = 22342
gi_GLA01_Land_ID = 19543
gi_GLA01_Ocean_ID = 20311
#
# Atmospheric Profile Parameters
#
gd_binsize = 76.8
#
# Elev Parameters
#
gd_ellipAe = 6378136.49
gd_ellipF = 298.25645
#
END_OF_CONSTANTS_MOD = ----------------------------------------------------
--
#
# GLAS_Error Mod Entries
#
BEG_OF_GLASError_MOD = ----------------------------------------------------
--
#
# Set Number of Errors
#
NUM_ERRORS=21
#
ERROR=10000   Error Opening File for Input:              3        1
ERROR=10100   Error Opening File for Output:             3        1
ERROR=10200   Error Closing File:                        3        1
ERROR=10300   Error Reading File:                        3        1
ERROR=10400   Error Writing File:                        3        1
ERROR=10700   Multiple single-instance keywords:         1        1
ERROR=10800   Multiple-instance keyword limit exceeded:  3        1
ERROR=10900   Unrecognized line in control file:         1        1
ERROR=11000   Unknown value in keyword/value pair:       1        1
ERROR=11100   I/O Error Opening Control File:            3        1
ERROR=11200   I/O Error Reading Control File:            3        1
ERROR=11300   Specified Unknown File Type:               1        1
ERROR=11400   GLA01 Unknown Record Type                  3        1
ERROR=20200   Error reading PAD data Eng data           3        2
ERROR=20304   Error reading PAD data Eng data           3        1
ERROR=30400   Error in Waveform Assess                  3        1
ERROR=35600   Error in Waveform Calc                    3        1
ERROR=45000   Error calculating backscatter            3        3
ERROR=49444   Error calculating Backscatter Profile     3        1
ERROR=50000   Error Calculating tides                   3        10
ERROR=59999   Error in Geoids module                    3        4
#
```

```
END_OF_GLASError_MOD = ------------------------------------------------------
--
#
END_OF_ANC07_FILE = ------------------------------------------------------
--
```

## Fortran/Sub-ATBD Interface

Definitions:

(1) algorithm data (units for algorithm use) are that data which are in a form most favorable for display and calculation;

(2) product data (units for I/O) are that data which are in a form most favorable for machine independence and storage efficiency.

A simplified, data-specific description of the processing flow from input SCF file to output SCF file follows:

(1) Data are stored in GLAS SCF files in a product form.

(2) The GLAS_Exec read routine will call a product-specific Fortran 90 routine. This routine calls the f90 intrinsic **fread** to read the product record. The product record is stored in a public structure in a product-specific module (GLAxx_mod.f90).

(3) The f90 product structure will then be converted into an f90 algorithm structure. The conversion is a simple data type and unit transformation – there is a one-to-one correspondence between the variables described in each product definition and the f90 algorithm structure.

(4) The resultant f90 algorithm structure is stored in a public structure in a product-specific module (GLAxx_mod.f90).

(5) All "pass-thru" data from input product(s) are copied to the output product.

(6) In a reprocessing scenario, when using same-type input and output products, data are copied from input product to output product.

(7) Required input and output data structures are accessed by the appropriate Subsystem Manager.

(8) Data which need further conversion in order to be usable by the sub-ATBDs are converted and placed in local variables. (This includes such conversions as unpacking flags and adding offsets or reference values).

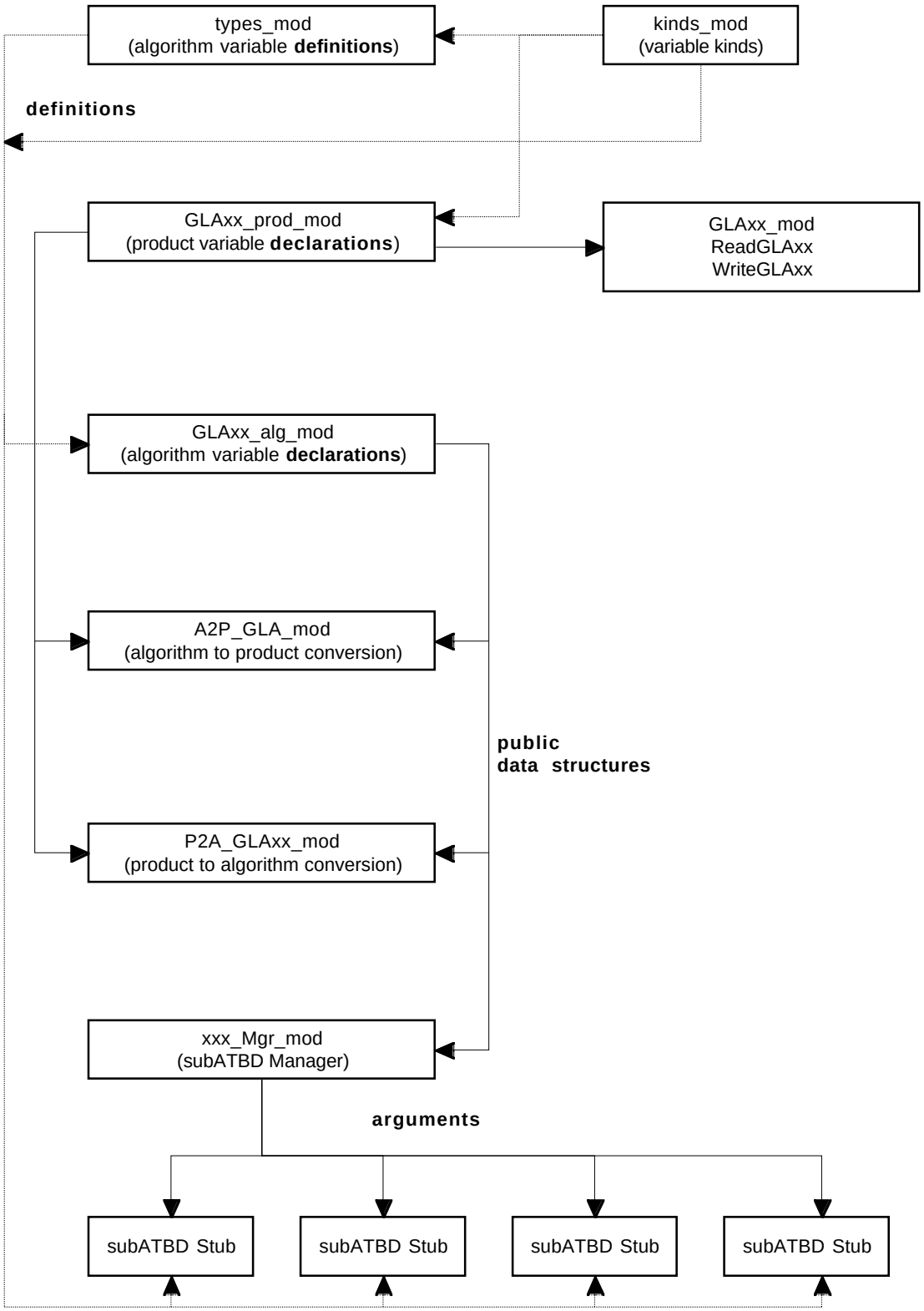
(9)The manager passes (via an argument list) data to/from each sub-ATBD. Sub-ATBDs only have access to the data they need, not the whole data structure. Additionally, data are passed primarily from the output structure. Those data not available from the output structure are passed from the input structure. This method of passing data, along with the processes described in steps 5 and 6, allow the Sub-ATBDs to be coded without regard to processing or reprocessing scenarios.

(10) Data are processed by the appropriate Sub-ATBDs.

(11) Local variables which need further conversion in order to be consistent with the output structure are converted. (This includes such conversions as packing flags and subtracting offsets or reference values).

(12) The algorithm f90 structure will be converted into a product f90 structure. The conversion is a simple data type and unit transformation – there is a one-to-one

correspondence between the variables described in each product definition and the f90 algorithm structure.
(13) The product f90 structure will be written by a product-specific Fortran90 routine using the **fwrite** intrinsic.
The benefit of this method is that the I/O and scaling routines are very language-independent and that scaling/unscaling is handled inside the I/O routines so that both internal and external programmers can simply call the I/O routine and end up with a structure of algorithm data.


types_mod
(algorithm variable **definitions**)
kinds_mod
(variable kinds)
**definitions**
GLAxx_prod_mod
(product variable **declarations**)
GLAxx_mod
ReadGLAxx
WriteGLAxx
GLAxx_alg_mod
(algorithm variable **declarations**)
A2P_GLA_mod
(algorithm to product conversion)
**public**
**data structures**
P2A_GLAxx_mod
(product to algorithm conversion)
xxx_Mgr_mod
(subATBD Manager)
**arguments**
subATBD Stub
subATBD Stub
subATBD Stub
subATBD Stub

## Fortran Subroutine Interface Mechanism

Subsystems and Managers use f90 modules to **define** data types. Modules do not create global data except in special circumstances dictated by GLAS_Exec. Structures are defined with the *type–endtype* construct and variables are defined with the same type of constructs used by *kindsmod*. The benefits of this are that even though variables are passed from Manager to ATBD via the argument list, the use of modules minimizes the risk of mismatched data types between the Manager and ATBD. We get the benefits of :

- Guaranteeing local variables via the argument list construct
- Ability to declare *INTENT(IN)* and INTENT(OUT)
- Ability to change the variable type or dimensions in one place and have this change automatically propagate to all routines which use the variable.
- Less chance of errors since data types are defined in only place.

## Source Files

Version 0 source files consist of a group of top level files which are grouped together with the main program GLAS_Exec.f90, and a lower level set of files grouped together by functionality and separated as libraries.

### Top Level files

The Top level files consist of the sub-Managers, the read and write modules and the wrap-up modules. They are located in the /src/GLAS_Exec directory and are the following:

| | | |
|---|---|---|
| GLAS_Exec.f90 | - | The main program |
| L1Amgr_mod.f90 | - | Sub-manager for L1A processing |
| WFMgr_mod.f90 | - | Sub-manager for waveform processing |
| AtmMgr_mod.f90 | - | Sub-manager for Atmosphere processing |
| ElevMgr_mod.f90 | - | Sub-manager for elevation processing |
| OpenFiles_mod.f90 | - | Opens the input files specified by the control file |
| ReadAnc_mod.f90 | - | Reads the ancillary file |
| Readdata_mod.f90 | - | Reads the input data |
| WriteL1A_mod.f90 | - | Writes products GLA01 – GLA04 |
| WriteWF_mod.f90 | - | Writes product GLA05 |
| WriteAtm_mod.f90 | - | Write product GLA06 – GLA11 |
| WriteElev_mod.f90 | - | Writes products GLA12 – GLA15 |
| CloseFiles_mod.f90 | - | Closes all files |
| MainWrap_mod.f90 | - | Writes summary data and completes processing |

A makefile is contained in this directory. It creates the executable for the software, linking in the libraries described below.

### Lower Level Files, Libraries

Libraries are the key to making GLAS_Exec and related I-SIPS software consistent and maintainable. Each library directory contains a makefile that creates the library. The libraries are in subdirectories under the /src (source) directory and are listed in the appendix. This section identifies each library and the contents thereof:

### libcio.a – Generic C routines for reading and writing data.

| | |
|---|---|
| cappend.c | positions file pointer to EOF on opened file. |
| cclose.c | closes and open file. |
| cnlines.c | counts number of lines in ASCII file. |
| cfsize.c | counts number of bytes in file. |
| copen.c | opens a file for specified activity. |
| cread.c | reads bytes from a file. |
| crewind.c | rewinds an open file. |
| cseek.c | positions file pointer on opened file |
| cwrite.c | writes bytes to a file. |
| vers_cio_mod.f90 | Version information for library |

### libprod.a –F90 routines for reading and/or writing each GLA/ANC file.

| | |
|---|---|
| GLAxx_prod_mod.f90 | GLA*XX* PRODUCT structures. |
| GLAxx_alg_mod.f90 | GLAxx ALGORITHM structures. |
| GLAxx_mod.f90 | reads and write GLA*XX* data. |
| GLAxx_scal_mod.f90 | Initializes GLAxx scales |
| conversion. | GLA*xxX* PRODUCT to ALGORITHM |
| conversion. | GLA*xx* ALGORITHM to PRODUCT |
| GLAxx_Pass_mod.f90 | Handles GLAxx Pass-Thru's. |
| vers_io_mod.f90 | Version information for library |

### libcntrl.a-f90 routines for control file and interface handling

| | |
|---|---|
| centertext_mod.f90 | Centers string on 80 column line. |
| doubleline_mod.f90 | Prints 80 column double line (=) |
| exec_flag_mod.f90 | Module which defines exec_flags |
| finfo_mod.f90 | Module which defines file control structures |
| getans.f90 | Gets single-key input from user |
| keyval_mod.f90 | Defines keyword-value data type |
| multimenu_mod.f90 | Gets user choices from a multiple-option menu |
| parse_keyval_mod.f90 | Parses keyword-value from a string |
| singleline_mod.f90 | Prints 80 column single line (-) |

| | |
|---|---|
| strtrim.f90 | Strips spaces from a string. |
| tolower.f90 | Converts string to lower case |
| toupper.f90 | Converts string to upper case |
| writebanner_mod.f90 | Writes informative banner |
| vers_cntrl_mod.f90 | Version information for library |

### libfile-Generic file routines

| | |
|---|---|
| OpenFInFile.f90 | Opens a Fortran90 style input file. |
| OpenFOutFile.f90 | Opens a Fortran90 style output file. |
| OpenCInFile.f90 | Opens a C style input file. |
| OpenCOutFile.f90 | Opens a C style output file. |
| vers_file_mod.f90 | Version information for library |

### libanc-Ancillary file routines

| | |
|---|---|
| ANC07_mod.f90 | ANC07 file reader. |
| vers_anc_mod.f90 | Version information for library |

### libtime-Misc time conversion routines

| | |
|---|---|
| sctime_2i4_to_d_mod.f90 | 2 i4b to dp |
| sctime_d_to_2i4_mod.f90 | dp to 2 i4b |
| sctime_i5_to_d_mod.f90 | i4 to dp |
| sctime_d_to_i5_mod.f90 | dp to i5. |
| vers_time_mod.f90 | Version information for library |

### liberr-Error handling routines

| | |
|---|---|
| GLASError_mod.f90 | Error Handling |
| ANC06_mod.f90 | ANC06 file writer. |
| vers_err_mod.f90 | Version information for library |

### libplatform.a-platform-specific routines

| | |
|---|---|
| kinds_mod.f90 | Kinds module |
| types_mod.f90 | Types module |
| constants_mod.f90 | Constants module. |
| vers_platform_mod.f90 | Version information for library |

### libl1a.a – L1A algorithm routines

| | |
|---|---|
| L_Alt.f90 | Calculates altitude products |
| L_Atm.f90 | Calculates atmosphere products |

### libatm.a - Atmosphere algorithm routines

| | |
|---|---|
| A_intrp_geoloc_mod.f90 | Interpolation for POD |
| A_interp_met_mod.f90 | Interpolation for MET |

| | |
|---|---|
| A_mbscs_mod.f90 | Calculates backscatter profiles |
| A_cal_cofs_mod.f90 | Calculates calibration coefficients |
| A_ir_bscs_mod.f90 | Calculates cross section profiles for 1064 |
| A_g_bscs_mod.f90 | Calculates cross section profiles for 532 |
| A_avg_bscs_mod.f90 | Calculates backscatter averages |
| A_cld_lays_mod.f90 | Calculates cloud layers |
| A_pbl_lay_mod.f90 | Calculates PBL layer |
| A_aer_lays_mod.f90 | Calculates elevated aerosol layers |
| A_aer_opt_prop_mod.f90 | Calculates aerosol optical properties |
| A_cld_opt_prop_mod.f90 | Calculates cloud optical properties |

### libelev.a – Elevation algorithm routines

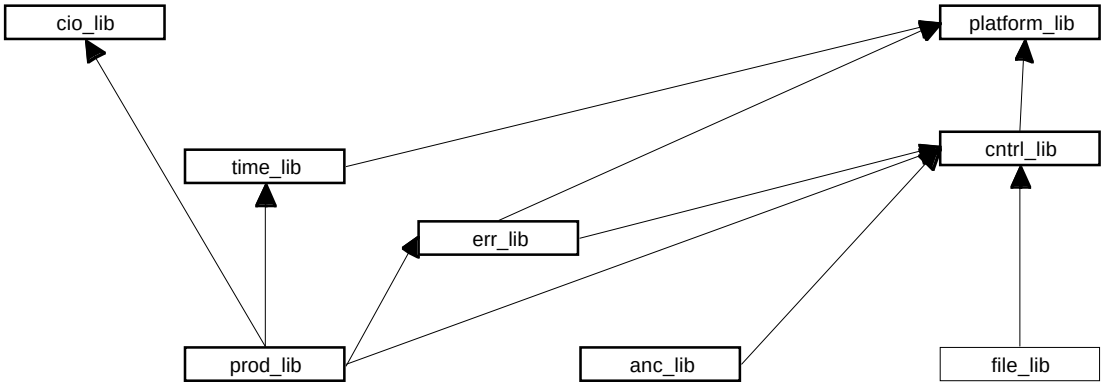
| | |
|---|---|
| E_calcLoadTD_mod.f90 | Calculates load tide |
| E_calcOceanTD_mod.f90 | Calculates ocean tide |
| E_calcEarthTD_mod.f90 | Calculates earth tide |
| E_calcPoleTD_mod.f90 | Calculates pole tide |
| E_getgeoid_mod.f90 | Calculates geoid height |
| E_calcTrop_mod.f90 | Calculates tropospheric correction |
| E_calcRange_mod.f90 | Calculates standard instrument range |
| E_calcRngOff_mod.f90 | Calculate instrument range offsets |
| C_intrpPod_mod.f90 | Interpolates for POD |
| C_calcSploc_mod.f90 | Calculates elevation and spot location |
| E_atmQF_mod.f90 | Calculates atmosphere quality flags |
| E_calcSlope_mod.f90 | Calculates slope and Roughness |
| E_calcRefl_mod.f90 | Calculates reflectance |
| E_chkReg_mod.f90 | Checks the region |
| E_calcRegParm_mod.f90 | Calculates region parameters |

### libwf.a – Waveform algorithm routines

| | |
|---|---|
| W_Assess_mod.f90 | Assessment of waveforms |
| W_CalcOtherCh_mod.f90 | Calculates other waveform characteristics |
| W_DetGeoSurfTyp_mod.f90 | Determines waveform geolocation and surface type |

## Library Dependencies

Since all of our Fortran code is developed with modules, there are specific dependency requirements imposed since the .mod files need to be linked at compile-time rather than link time. The following diagram shows the library dependencies.



# Appendix

## Directory structure (Top Level files and Libraries)

The main directories where source files and libraries are located are listed below.

/src

- GLAS_Exec (Top level files)
- L1a_lib (Submanager library)
- Atm_lib "
- Wf_lib "
- Elev_lib "
- L1a (Level 1A source)
- Waveforms (waveform source)
- Atmosphere (atmosphere source)
- Elevations (elevation source)
- Modules (Fortran modules)
- Common_libs
  - Anc_lib
  - Cio_lib
  - Cntrl_lib
  - Err_lib
  - File_lib
  - Platform_lib
  - Prod_lib
  - Time_lib